第２章 アクティブ・アンテナの基本動作を理解し HDLで記述する

ディジタル・ビーム形成受信機のプロトタイプ設計

Minseok Kim

第１章ではアダプティブ・アンテナの概要と最近の移動通信システムの動向について説明した．本章ではアダプティブ・アレイの基本動作であるディジタル・ビーム形成法の原理について説明し，プロトタイプのハードウェアの設計について紹介する．（編集部）

1．ディジタル・ビーム形成（Digital Beamforming：DBF）

移動通信におけるデータ容量の増大やデータそのものの高品質化，広帯域データ伝送への要望が高まるなか，既存の周波数の利用効率を向上させることは，今後は欠かせない要求項目になってきます．

アレイ・アンテナを用いて，空間的に方向性を制御（指向性制御）すれば，所望の方向に送信電力を集中させ，無駄の少ない通信を行えます．送信電力の無駄な放射が抑えられるため，消費電力の低減や隣接のチャネルに影響する干渉電力の制御も可能になり，結果的に周波数の利用効率を向上することができます．

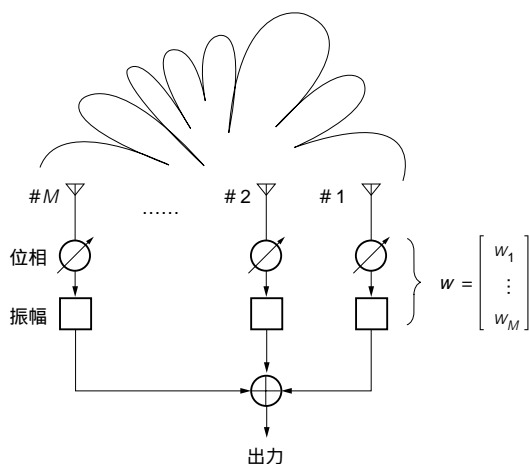



図1　ビーム形成の概念
各アンテナに到達する信号の位相と振幅をそろえて合成する

このようなアレイ・アンテナ信号処理技術を「ビーム形成（Beamforming）」といいます．特にアナログ制御ではなく，ディジタル信号処理により行う場合には「ディジタル・ビーム形成（Digital Beamforming）」といいます．

図1にアレイ・アンテナ・システムの概要を示します．M素子のアレイ・アンテナに受信されるそれぞれの信号に位相と振幅の制御を行い，信号対ノイズ比が最大になるように合成することになります．各アンテナ素子は固定的な放射指向性特性をもちますが，ここで位相と振幅の適応制御により仮想的に任意の受信ビームを形成できます．

これをアナログ回路で実現することも考えられますが，**図2**のようにA-Dコンバータを用いてアナログ信号をディジタル信号に変換し，ディジタル信号処理器の算術演算により実現することもできます．ディジタル信号処理器は汎用のDSP（Digital Signal Processor）あるいは高速で専用の機能が実現できるFPGA（Field Programmable Gate Array）が用いられます．ここでは，ディジタル・ビーム形成法について解説します．

● アレイ・アンテナの基本原理

原理を説明するために，**図3**のような狭帯域の簡単な平面波モデルを考えます(1)．アレイ・アンテナの構造は半波

KeyWord　ディジタル・ビーム形成，指向性制御，マルチパス，ディジタル受信機，4倍オーバサンプリング

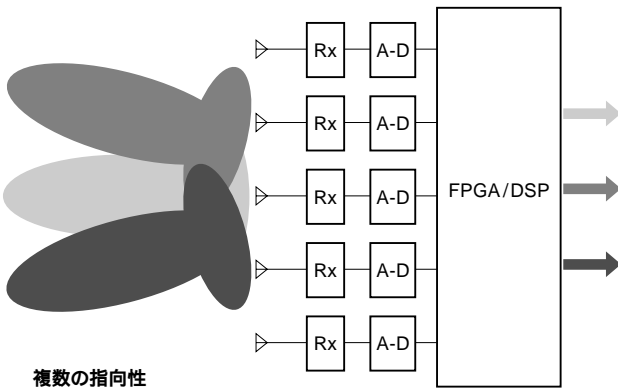
複数の指向性

Rx：Receiver
A-D：A-Dコンバータ

図2　ディジタル・ビーム形成システム

図3　平面波モデルにおける各受信アンテナにおける経路差

長の等間隔に配置したM個のリニア(線形)アンテナ素子とします．各アンテナは無指向性(Omni-directional)で，それぞれ同じ特性を持つことを前提に話を進めます．電波信号が，ある方向θから入射したとき，k番目のアンテナ素子における受信電圧は，基準アンテナにおける電圧から以下のように表現されます．

$$x_k(t) = x_0(t-\tau_k) \quad \cdots\cdots (1)$$

ここで，$\tau_k = \{d(k-1)\sin\theta\}/c$は到達時間差です．θは到来角(Direction of Arrival：DOA)，cは光速，dは素子間隔です．ただし，その信号が十分に狭帯域な信号の場合，各アンテナ素子に到達する時間差は，位相差($e^{-j\Phi}$)に置き換えて考えられ，以下のように表現できます．

$$x_k(t) = s(t)\exp\left(-j\frac{2\pi}{\lambda}d(k-1)\sin\theta\right)+n_k(t) \quad \cdots\cdots (2)$$
$$k=1,2,\cdots,M$$

ここで，$s(t)$は入射波の包絡線，λは波長，n_kは白色ガウスノイズを意味します．各アンテナでの受信信号に対して，位相と振幅の制御を行い，その結果を合成するビーム形成の動作は，次式のように表されます．

$$y(t) = \sum_{k=1}^{M}\omega_k^* x_k(t) \quad \cdots\cdots (3)$$

ここで，ω_kはk素子目の位相や振幅を示す値(重み係数値)で，一般に複素数となります．記号＊は複素共役を意味します．ベクトル表現を使えば以下のように簡単に表現できます．

$$y(t) = \boldsymbol{\omega}^{\mathrm{H}}\boldsymbol{x}(t) \quad \cdots\cdots (4)$$

ここで記号Hは複素共役転置を意味します．また，

$$\boldsymbol{\omega} = [\omega_1 \cdots \omega_M]^{\mathrm{T}}$$
$$\boldsymbol{x}(t) = \boldsymbol{a}(\theta)\boldsymbol{s}(t)+\boldsymbol{n}(t) \quad \cdots\cdots (5)$$

$\boldsymbol{a}(\theta)$は方向を特定するベクトルで，方向ベクトルあるいはステアリング・ベクトルと呼ばれます．

$$\boldsymbol{a}(\theta) = \left[1, \exp\left(-j\frac{2\pi}{\lambda}d\sin\theta\right), \cdots, \exp\left(-j\frac{2\pi}{\lambda}d(M-1)\sin\theta\right)\right]^{\mathrm{T}} \quad \cdots\cdots (6)$$

入射信号が平面波として到来することはあくまで理想的な仮定です．実際には見通しの環境は少なく，受信アンテナには多数のマルチパス波からなる合成信号として受信されます．N個の信号がそれぞれ$L(t)$個のマルチパス波を持つ場合について，もう少し一般的に表現したのが次の式です．

$$\boldsymbol{x}(t) = \sum_{i=0}^{N-1}\boldsymbol{v}_i(t,\tau)s_i(t)+\boldsymbol{n}(t) \quad \cdots\cdots (7)$$

ここで，$\boldsymbol{v}_i(\theta)$はチャネル応答ベクトルと呼ばれ，以下のような関係として表されます．

$$\boldsymbol{v}_i(t,\tau) = \sum_{l=0}^{L(t)-1}A_{l,i}(t)e^{j\phi_{l,i}(t)}\boldsymbol{a}(\theta_{l,i}(t))\delta(t-\tau_{l,i}(t)) \quad \cdots\cdots (8)$$

$A_{l,i}$，$\phi_{l,i}$，$\tau_{l,i}$，$\theta_{l,i}$は，信号iによる受信信号成分lの振幅，搬送波位相シフト，遅延量，到来方向です．

ディジタル・ビーム形成において，式(4)のようにウェイト・ベクトル$\boldsymbol{\omega}$をうまく制御することによって受信信号の品質(信号対雑音比)を改善できます．原理的には，M個の素子を使って合成する場合に10 log$M$の利得が得られます．例えば4素子の場合，10 log4　6 dBの利得になります．

参考として**図4**に素子数による合成利得を示します．ここで，1素子の場合は無指向性であり，どの方向も同じレ

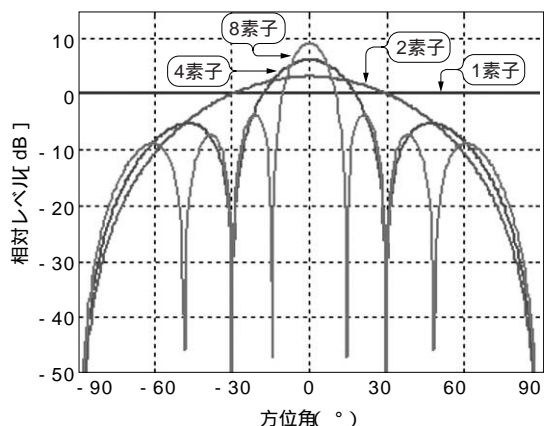



図4 ディジタル・ビーム形成の利得
素子数が増えるほどビームが鋭くなるだけでなくヌル点も多く形成される.

ベルで受信しますが，複数のアンテナで受信し一様に合成することで，ある特定方向にビームが形成されます．さらに素子数が増えるほどビームの幅が鋭くなり，利得も向上することが分かります．また，ビームだけでなくヌル点も形成されます．このようにビームとヌル点を適切に制御するのが後述のアダプティブ・アレイの基本動作原理です．

● DBF評価用プロトタイプ・ハードウェア

アレイ・アンテナ信号処理の評価のために設計した，プロトタイプのハードウェアは，**図5**のようにアレイ・アンテナ，RF受信機，A-D変換部，信号処理部，制御部で構成されます(5)．

アレイ・アンテナは任意の形状で16素子まで収容できます．RF受信機はYIG（Yttrium-Iron-Garnet）バンドパス・フィルタを用いて，移動通信システムで多く使われる周波数である2G～5GHzまで対応できるようになっています．RF受信機の出力は40MHzのIF（Intermediate Freguency）信号になり，後段のA-Dコンバータは32MHzでアンダサンプリングを行います．サンプリングされた信号はFPGAで準同期検波により複素ベースバンド信号に変換されます．出力信号は各信号に重み係数を乗算することにより最適な値として合成されます．

A-D変換ボード上には，約100万ゲート相当の大容量FPGA（Stratix「EP1S40」，米国Altera社）が搭載されており，高速の適応信号処理を行うことができます．制御ボードは，CPU（SH4，ルネサス・テクノロジ）の上にOS（NetBSD）が組み込まれており，システムの制御やモニタはEthernet経由で行われます(4)．

写真1～**写真3**に各部の様子を示します．また，ディジタル信号処理部の仕様を**表1**に示します．さらに**図6**に信号処理結果のGUI表示例を紹介します．このような評価システムを構築することで，さまざまな信号処理を柔軟に試すことができます．

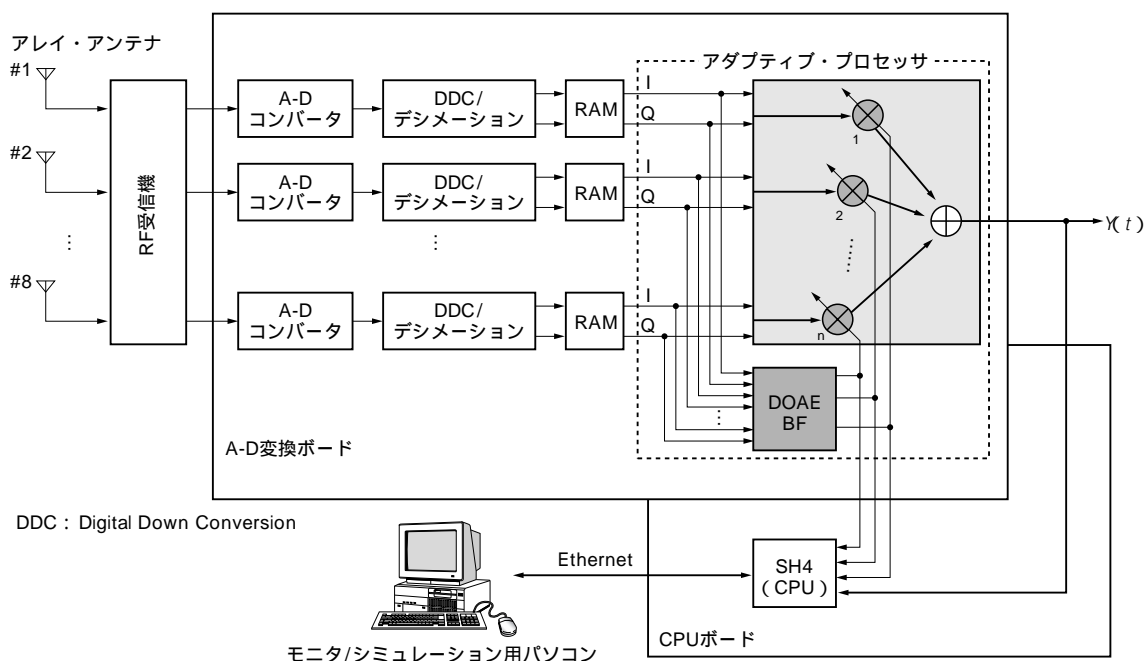



図5 DBFプロトタイプのハードウェアのブロック図

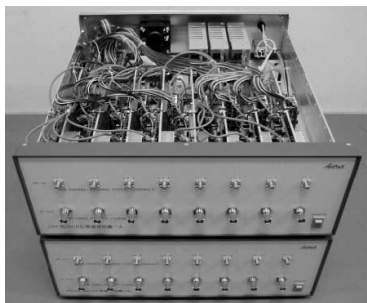



写真1　DBF受信機（RF部）
横浜国立大学新井研究室提供．

写真2　8素子リニア・アレイ・アンテナ
横浜国立大学新井研究室提供．

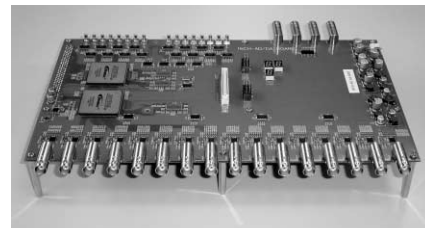



写真3　DBF受信機（ディジタル部）
横浜国立大学新井研究室提供．

2．DBF受信機の設計

● ディジタル受信機の構成

ディジタル技術の発展にともない，無線機における周波数のアップ/ダウン・コンバージョン，ミキサ，フィルタ，ゲイン調整，変復調といったアナログ技術で実現されていた多くの機能が，ディジタル信号処理に置き換えられる時代になってきました．近年，リコンフィギャラブル・デバイスを用いて，一つの端末機より回路の変更なしにさまざ

表1　ディジタル信号処理部の仕様

A-Dコンバータ	型番	AD9245（米国Analog Devices社）
	チャネル数	16
	分解能	14ビット
	サンプル・レート	80 MSPS
バッファ・メモリ	2Kワード/チャネル（FPGA内部メモリ 512Mバイト SDRAM）	
D-Aコンバータ	型番	MAX5195（米国Maxim Integrated Products社）
	チャネル数	16
	分解能	14ビット
	サンプル・レート	260 MSPS
FPGA	型番	Stratix EP1S40（米国Altera社）
	機能	- 41,250 LE（約1,000,000ゲート） - 3,423,744内部メモリ・ビット - 14 DSPブロック（14並列36×36乗算器実装可能）
CPU	型番	SH4
	性能	200 MHz, 360 MIPS, 1.4 GFLOPS
	OS	NetBSD1.5（NetBSD/SH3）
ユーザ・インターフェース	Ethernet	100 Base-T

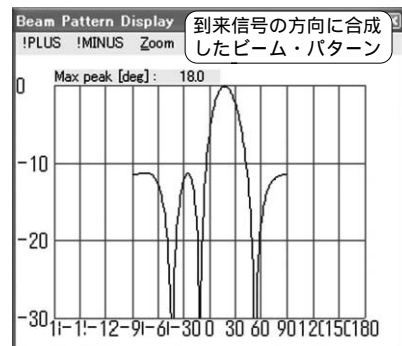
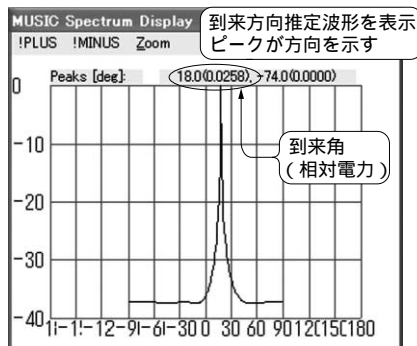
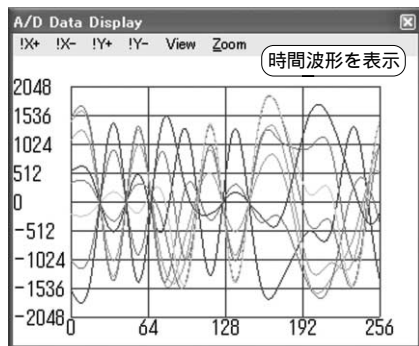
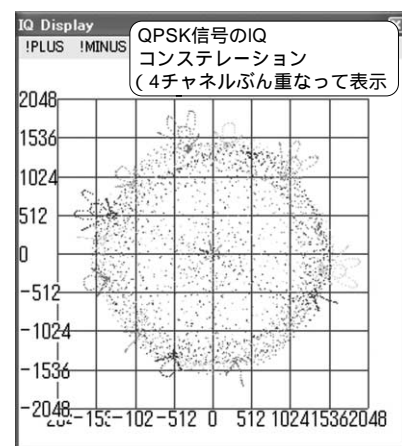



図6　計算結果のGUI実時間表示例
横浜国立大学新井研究室提供．

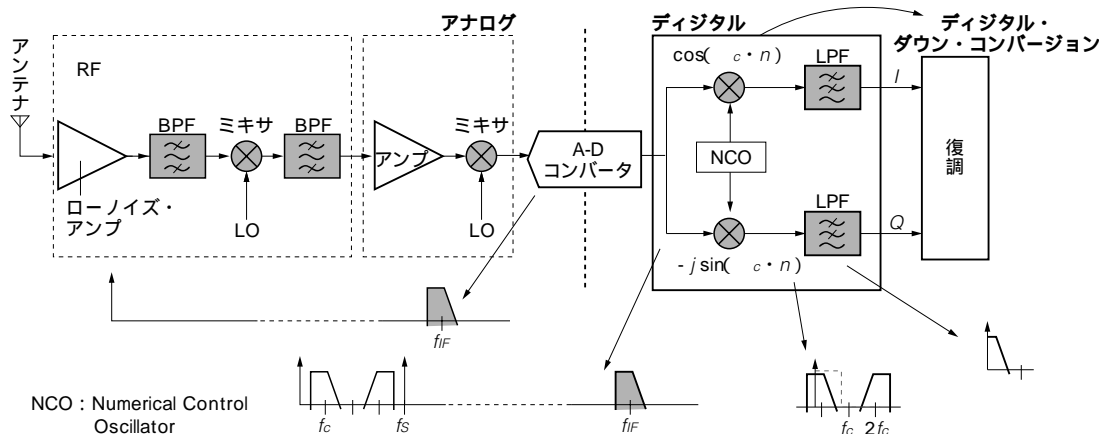



図7 ディジタル受信機の構成

NCO：Numerical Control Oscillator

まな機能追加やシステム変更に対応するソフトウェア無線（Software Defined Radio）技術が注目されています．特性の変更が困難なアナログ回路をディジタル回路化することは，システムの柔軟性を高める有効な手段になります(2)．

受信機においては，以前からさまざまなアーキテクチャが工夫されており，中間周波数を用いるスーパヘテロダイン型注1がもっとも一般的に使われています．また，ディジタル受信機の場合，A-Dコンバータの位置がシステム構造を左右する主な要因になります．

現在，通信用A-Dコンバータのサンプリング・レートは数十～数百MHzのものが一般的であり，IF周波数を直接サンプリングすることができます．これをIFサンプリング方式といいます．サンプリングされたIF信号はディジタル・ダウン・コンバージョンにより，複素ベースバンド信号に変換されます．ディジタル・ダウン・コンバージョンはローカル発振器をディジタル化した数値制御発振器（NCO）と乗算器，FIRフィルタで構成します．

信号周波数より低い周波数でのサンプリングとして，アンダーサンプリング（バンドパス・サンプリング）という手法があります．これはRF信号をアナログ信号処理によって，いったんある程度高いIF周波数に変換し，IFより低い周波数でA-D変換を行います．この場合，サンプリング・レートが，それほど高速である必要はないことから，低消費電力，高分解能のA-Dコンバータを用いることができます．アナログ信号処理部においては，Q値の高いフィルタなど大型で特性の変更が難しい素子の使用が不可欠となります．また，IFの利用により生じるイメージ信号の抑圧のため，フィルタなどの処理が必要となることや，IF周波数がサンプリング・レートより高ければ高いほどクロック・ジッタに影響されやすいことなどが，欠点として挙げられます(3)．しかし，現時点で有効な手法として広く用いられます．

注1：搬送周波数とローカルに生成される周波数を，ミキサを用いて低周波数信号（中間周波数）に変換する受信機．低周波数信号とすると，元の変調搬送波よりも安定であり，復調するのが容易である．

● 準同期検波を用いるディジタル受信機の設計

図7に今回用いるディジタル受信機の構成を示します．信号スペクトルがRFからベースバンドに変換されていく様子を表しています．ここでは，簡単のため搬送波再生を必要としない準同期検波方式のディジタル受信機の設計について説明します．

IFサンプリングを用いることで，ベースバンド・サンプリングよりもA-Dコンバータの数を半分に減らすことができ，アレイ・アンテナ信号処理の低コスト化に繋がります．位相変調（PSK：Phase Shift Keying）された信号の場合，サンプリングされたIF信号は，以下のように表現できます．

$$x(n) = x_I(n)\cos\omega_c n + x_Q(n)\sin\omega_c n \quad \cdots\cdots\cdots\cdots\cdots\cdots(9)$$

ここで，$x_I(n)$と$x_Q(n)$はIF信号$x(n)$の同相成分と直交成分になり，ω_cは搬送波の角周波数を表します．ダウン・コンバージョンは以下の式のように，IF信号に角周波数ω_cの複素発振信号（NCO）を乗算します．

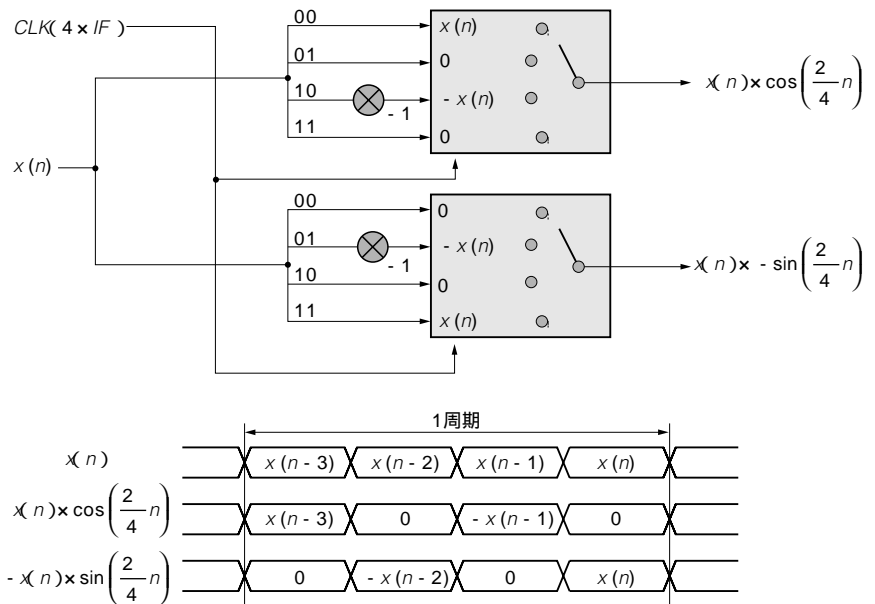
$$\begin{aligned}\tilde{x}(n) &= x(n)[\cos\omega_c n - j\sin\omega_c n] \\ &= \frac{1}{2}[x_I(n) + x_I(n)\cos 2\omega_c n - jx_I(n)\sin 2\omega_c n \quad \cdots\cdots(10) \\ &\quad - jx_Q(n) + x_Q(n)\sin 2\omega_c n + jx_Q(n)\cos 2\omega_c n]\end{aligned}$$

そうすると，直流成分（DC）とω_cの2倍成分がそれぞれ

cos (2πfc・n)
LPF
I ベースバンド信号 (同相成分)
IF信号 x(n)
A-D コンバータ
NCO
LPF：Lowpass Filter
NCO：Numerical Control Oscillator (数値制御発信器)
LPF
Q ベースバンド信号 (直交成分)
−j sin (2πfc・n)
アナログ信号 (中心周波数fc)
ナイキスト領域
2倍成分
低域通過フィルタ
2倍成分はフィルタにより除去される
fc
fs/2

図8
ディジタル・ダウン・コンバージョンの動作

1
1 App
2
2 App
3
3 App
4

図9
4倍オーバ・サンプリングを用いたNCOとミキサの回路

現れます．低域通過フィルタを用いて2倍の高周波成分を取り除くことで，以下のように結局IF信号からω_cだけシフトされた複素ベースバンド信号が得られます．

$$\mathrm{LPF}(\tilde{x}(n)) = \frac{1}{2}[x_I(n) - jx_Q(n)] \quad \cdots\cdots\cdots (11)$$

この動作を詳細に示したのが**図8**です．

ここで，特にサンプリング周波数がIF搬送波周波数より4倍になる場合に，NCOと乗算器は単純化でき，**図9**のように入力信号をただ順にスイッチングするだけで実現できます．

低域通過フィルタについては，NCOとの乗算の結果から生じる2倍の信号成分を取り除くことのみを考慮し，簡単

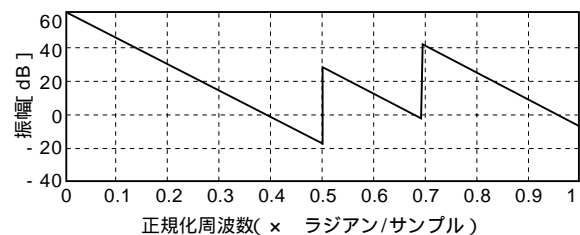
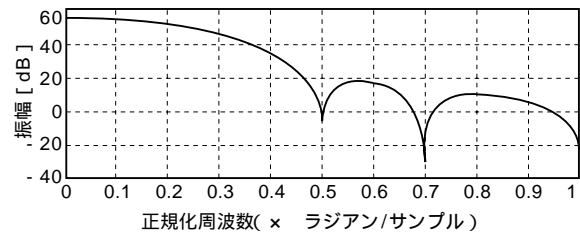



図10　低域通過フィルタの周波数特性

リスト1　ディジタル・ダウン・コンバージョン(DDC)の実装例

```
=======================================================
--  DDC (Digital Down Conversion & I/Q detection)
--  Simple NCO : 4 value switching circuit
--
=======================================================

LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_arith.ALL;
USE IEEE.std_logic_UNSIGNED.all;
USE work.util_package.ALL;

ENTITY ddc12bit IS
  PORT (
      clk      : IN std_logic;
      nrst     : IN std_logic;
      nbypass  : IN std_logic;
      ena      : IN std_logic;
      sintab   : IN std_logic_vector(1 downto 0);
      x_in     : IN std_logic_vector(11 downto 0);
      i_out    : OUT std_logic_vector(11 downto 0);
      q_out    : OUT std_logic_vector(11 downto 0));
END ENTITY ddc12bit;

ARCHITECTURE rtl OF ddc12bit IS

  COMPONENT fir12x8 IS
    PORT (
      clk      : IN std_logic;
      nrst     : IN std_logic;
      ena      : IN std_logic;

      x_in     : IN std_logic_vector(11 downto 0);
      y_out    : OUT std_logic_vector(11 downto 0));
  END COMPONENT fir12x8;

  COMPONENT fir12x8 IS
    PORT (
      clk      : IN std_logic;
      nrst     : IN std_logic;
      ena      : IN std_logic;

      x_in     : IN std_logic_vector(11 downto 0);
      y_out    : OUT std_logic_vector(11 downto 0));
  END COMPONENT fir12x8;

  SIGNAL  x_signed, x_cos, x_sin, i, q
                         : std_logic_vector(11 downto 0);
  SIGNAL  i_out_buff, q_out_buff
                         : std_logic_vector(11 downto 0);

  BEGIN

--  入力レジスタ，2の補数表現
    PROCESS (clk) IS
     BEGIN
       IF clk'event AND clk = '0' THEN
         IF ena = '1' THEN
           x_signed <= NOT x_in(11) & x_in(10 downto 0);
         END IF;
       END IF;
      END PROCESS;

--  4倍オーバサンプリングのNCO
    x_cos <= x_signed          WHEN sintab = "00" ELSE
             (OTHERS=>'0')     WHEN sintab = "01" ELSE
             (NOT x_signed)+1  WHEN sintab = "10" ELSE
             (OTHERS=>'0')     WHEN sintab = "11" ELSE
             (OTHERS=>'0');

    x_sin <= (OTHERS=>'0')     WHEN sintab = "00" ELSE
             (NOT x_signed)+1  WHEN sintab = "01" ELSE
             (OTHERS=>'0')     WHEN sintab = "10" ELSE
             x_signed          WHEN sintab = "11" ELSE
             (OTHERS=>'0');

   --  低域通過フィルタで2倍の周波数成分を除去
    fir_ad_I : fir12x8 PORT MAP (clk, '1', '1', x_cos,
                                                      i);
    fir_ad_Q : fir12x8 PORT MAP (clk, '1', '1', x_sin,
                                                      q);

    -- ストレート・バイナリに変換
    i_out <=  NOT i(11) & i(10 downto 0) WHEN nbypass =
                                               '1' ELSE
              x_in;
    q_out <=  NOT q(11) & q(10 downto 0) WHEN nbypass =
                                               '1' ELSE
              (OTHERS=>'0');
  END;
--
=======================================================
```

のために8タップのFIR(Finite Impulse Response)フィルタで実装しました．その特性を**図10**に示します．A-Dコンバータから変換されたディジタル・データに直流のバイアスが乗っている場合，IF信号周波数領域に直流成分が，DDC(Digital Down Conversion)によりシフトされて現れます．これはベースバンド信号に高周波ノイズとして影響するので，低域通過特性とともにスペクトル中心にヌルが形成されるようにしてあります．

リスト1に，ここで説明したディジタル・ダウン・コンバージョン(DDC)回路のVHDL記述を紹介します．

参考・引用*文献

(1) 菊間信良；アダプティブアンテナ技術，2003年，オーム社.
(2) 原田博司；ソフトウェア無線をFPGAで実現する，Design Wave Magazine，2005年2月号，pp.28-104.
(3) M. Kim, A. Kiyono, K. Ichige, H. Arai；“*Experimental Study of Jitter Effect on Digital Downconversion Receiver with Undersampling Scheme,*” IEICE Transactions on Information and System, Vol. E88-D, No.7, pp. 1430 -1436, Jul. 2005.
(4) 堀内岳人；組み込み分野へのBSDの適用，Interface，第6章，pp.101-110，2002年8月号.
(5) Minseok Kim, Koichi Ichige and Hiroyuki Arai；“*16-element DOA estimation system,*” IEICE Technical Report, SR2005-43, YRP, Japan, July 2005.

Minseok Kim
東京工業大学

第2章 Appendix

FPGAのLUTを用いるFIRフィルタの実装

Minseok Kim

FIR(Finite Impulse Response Filter)フィルタとは，有限のインパルス応答で表現されるフィルタのことです．ディジタル信号処理には欠かせないものです．一般に次式のように表現されます．

$$y(n) = \sum_{i=l}^{N} h_i \cdot x(n-i)$$

ここで，hはフィルタ係数，xは入力信号です．FPGAを用いて実装するには乗算器をパラレルに構成すれば簡単に実現できます．また，最近のFPGAではFIRフィルタ専用のブロックを用意しているので，数百MHzの高速な性能を手軽に達成できるようになっています．しかし，ここではFPGAのLUT(Look-up Table)を用いて乗算器を必要としない構成について紹介します．

例えば，一般的な8タップFIRフィルタの場合には八つのフィルタ係数をもち，次式のように出力が計算されます．

$$y(n) = x(n)h_1 + x(n-1)h_2 + x(n-2)h_3 + x(n-3)h_4 \\ + x(n-4)h_5 + x(n-5)h_6 + x(n-6)h_7 + x(n-7)h_8$$

これは**図1**のような構造になります．もしフィルタ係数において左右対象性があれば，**図2**のように折り返し構造が使えるため，必要な乗算器の数を半分に減らせます．さらにLUTを用いれば，乗算器をまったく使わずに実装できます．

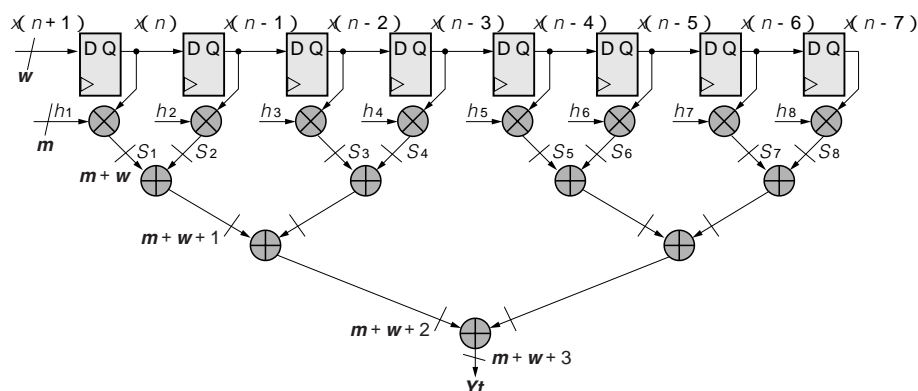



図1 FIRフィルタの一般的な構成

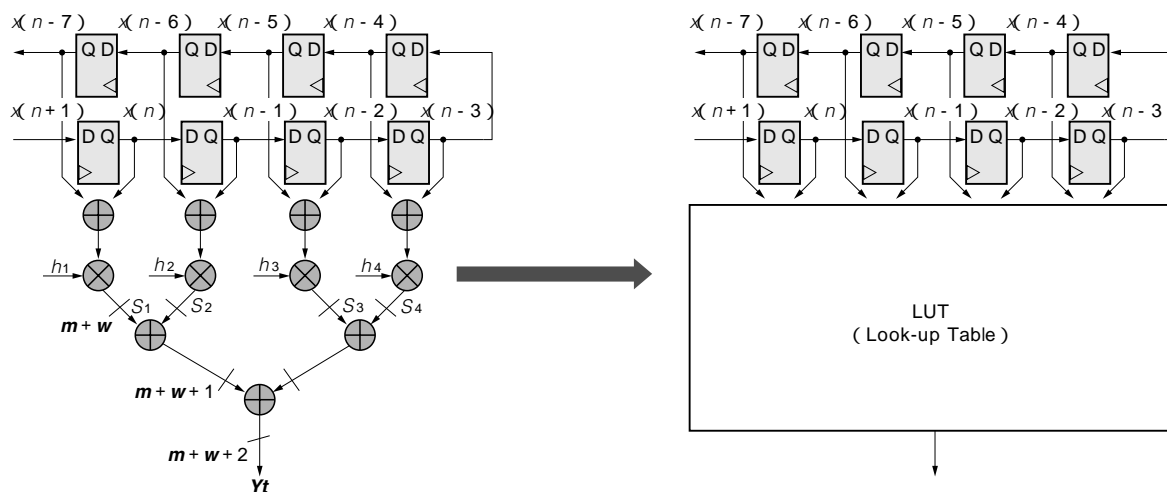



図2 フィルタ係数の対称性を用いた構成

```
hn  = 01  11  10  11 ──────→ hnは定数
×xn = 11  00  10  01
────────────────────────
p0  = 01  00  00  11  = 100
p1  = 01  00  10  00  = 011
────────────────────────
yn  = 011 000 100 011 = 1010
```

x_i^k		LUT量	x_i^k		LUT量
0000	=>	00+00+00+00 = 0000	1000	=>	01+00+00+00 = 0010
0001	=>	00+00+00+11 = 0011	1001	=>	01+00+00+11 = 0100
0010	=>	00+00+10+00 = 0010	1010	=>	01+00+10+00 = 0011
0011	=>	00+00+10+11 = 0101	1011	=>	01+00+10+11 = 0110
0100	=>	00+11+00+00 = 0011	1100	=>	01+11+00+00 = 0100
0101	=>	00+11+00+11 = 0110	1101	=>	01+11+00+11 = 0111
0110	=>	00+11+10+00 = 0101	1110	=>	01+11+10+00 = 0110
0111	=>	00+11+10+11 = 1000	1111	=>	01+11+10+11 = 1001

図3
4入力2ビットのLUTベースの乗算例

リスト1　LUTベースの低域通過フィルタのVHDL記述例

```
--*************************************--
--    FIR FILTER (LPF) USING LUT
--          (fir.vhd)
--       by Minseok Kim
--=====================================--
--  FEATURES
--  Using simplified LUT
--  Searching LUTDATA
--  Signed Input can be supported
--  Coefficient :  8 bits
--  Input       : 12 bits
--  Output      : 12 bits (Scaled)
--*************************************--

LIBRARY ieee;
USE  ieee.std_logic_1164.all;
USE IEEE.std_logic_arith.all;
USE IEEE.std_logic_signed.all;

LIBRARY ieee;
USE  ieee.std_logic_1164.all;

PACKAGE util_package IS

  FUNCTION fixsub(x:std_logic_vector;
            y:std_logic_vector) RETURN std_logic_vector;
  FUNCTION fixadd(x:std_logic_vector;
            y:std_logic_vector) RETURN std_logic_vector;
  FUNCTION fixmult(x:std_logic_vector;
            y:std_logic_vector; n:natural)
          RETURN std_logic_vector;
END  util_package;

LIBRARY ieee;

PACKAGE BODY util_package IS

  FUNCTION fixadd(x:std_logic_vector; y:std_logic_vector)
    RETURN std_logic_vector IS
    VARIABLE z : std_logic_vector(x'length downto 0);
    BEGIN
      z := (x(x'high) & x) + (y(y'high) & y);
      RETURN z(x'length downto 1);
    END;

  FUNCTION fixadd(x:std_logic_vector;
                                y:std_logic_vector)
    RETURN std_logic_vector IS
    VARIABLE z : std_logic_vector(x'length downto 0);
    BEGIN
      z := (x(x'high) & x) + (y(y'high) & y);
      RETURN z(x'length downto 1);
    END;

  FUNCTION fixmult(x:std_logic_vector;
                     y:std_logic_vector; n:natural)
    RETURN std_logic_vector IS
    VARIABLE z     : std_logic_vector(x'length+y'length-
                                                1 downto 0);
    BEGIN
      z := Conv_std_logic_vector(Conv_integer(x)
                  *Conv_integer(y), x'length+y'length);
RETURN z(z'high downto (z'length-n));
    END;

END util_package;


LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_ARITH.all;
USE IEEE.std_logic_UNSIGNED.all;
USE work.util_package.ALL;

ENTITY fir12x8 IS
  PORT (
    clk      : IN std_logic;
    nrst     : IN std_logic;
    ena      : IN std_logic;
    x_in     : IN std_logic_vector(11 downto 0);
    y_out    : OUT std_logic_vector(11 downto 0));
END ENTITY fir12x8;

ARCHITECTURE rtl OF fir12x8 IS

-- coef  = [10 45 92 127 127 92 45 10];
  TYPE LUTFIR0 IS ARRAY(0 to 15) OF integer RANGE  0 to 511;
  CONSTANT LUT:LUTFIR0 :=
  (0,  10,   45,   55,   92,   102,  137,  147,
 127,  137,  172,  182,  219,  229,  264,  274);

  TYPE SR12 IS ARRAY (0 to 7) OF std_logic_vector
                                         (12 downto 0);
  SIGNAL tap_sr : SR12;

  TYPE SR12h IS ARRAY (0 to 3) OF std_logic_vector
                                         (12 downto 0);
  SIGNAL tapplus : SR12h;

  TYPE ADDR8 IS ARRAY (0 to 12) OF std_logic_vector
                                          (3 downto 0);
  SIGNAL addr : ADDR8;

  TYPE LD10 IS ARRAY (0 to 12) OF std_logic_vector
                                          (8 downto 0);
  SIGNAL p : LD10;

--scaling at last stage---------------------------------
  SIGNAL add10, add11, add12        :
std_logic_vector(10 downto 0);
  SIGNAL add13, add14, add15        :
std_logic_vector(10 downto 0);
  SIGNAL add20, add21, add22        :
std_logic_vector(13 downto 0);
  SIGNAL add30                      :
std_logic_vector(18 downto 0);
```

リスト1　LUTベースの低域通過フィルタのVHDL記述例(つづき)

```
  SIGNAL add31                     :
std_logic_vector(15 downto 0);
  SIGNAL yout                      :
std_logic_vector(24 downto 0);
-----------------------------------------------------

  BEGIN

    PROCESS (clk, nrst) IS
      BEGIN
        IF nrst = '0' THEN
          tap_sr(0 to 7) <= (OTHERS => "0100000000000");
        ELSIF clk'event AND clk = '1' THEN
          IF ena = '1' THEN
            tap_sr(1 to 7) <= tap_sr(0 to 6);
            -- 入力データを正数にするため
            tap_sr(0) <= (x_in(11) & x_in) +
                                              "0100000000000";
          END IF;
        END IF;
      END PROCESS;

loop0: FOR i IN 0 TO 3 GENERATE
     tapplus(i) <= fixadd(tap_sr(i),tap_sr(7-i));
END GENERATE;

loop1: FOR i IN 0 TO 12 GENERATE
         addr(i) <= tapplus(3)(i) & tapplus(2)(i) &
                    tapplus(1)(i) & tapplus(0)(i) ;
       END GENERATE;

-- p's are 10 bits unsigned
loop2: FOR i IN 0 TO 12 GENERATE
         p(i) <= Conv_std_logic_vector(
             LUT(Conv_integer(unsigned(addr(12-i)))), 9);
       END GENERATE;
--scaling at last stage--------------------------------
  PROCESS (clk) IS
    BEGIN
      IF clk'event AND clk = '1' THEN
        add10 <= ( "00" & p(1))  + ( '0'  & p(0)  & '0' );
  -- 11 bits
        add11 <= ( "00" & p(3))  + ( '0'  & p(2)  & '0' );
        add12 <= ( "00" & p(5))  + ( '0'  & p(4)  & '0' );
        add13 <= ( "00" & p(7))  + ( '0'  & p(6)  & '0' );
        add14 <= ( "00" & p(9))  + ( '0'  & p(8)  & '0' );
        add15 <= ( "00" & p(11)) + ( '0'  & p(10) & '0' );

        add20 <= ( "000" & add11) + ( '0' & add10 & "00" );
  -- 14 bits
        add21 <= ( "000" & add13) + ( '0' & add12 & "00" );
        add22 <= ( "000" & add15) + ( '0' & add14 & "00" );

        add30 <= ( "00000" & add21) + ( '0' & add20 & "0000" );
  -- 19 bits
        add31 <= ( "0000"  & p(12)) + ( '0' & add22 & '0' );
  -- 16 bits

        yout  <= (( "000000000" & add31) + ( '0' & add30
                                                & "00000" ))
                - "00000"&LUT(15)&"00000000000"
                                          ;  -- 25 bits (274)
    END IF;
  END PROCESS;
-----------------------------------------------------

    y_out <= Conv_std_logic_vector( Conv_integer(yout) /
2**7, 12 );

  END ARCHITECTURE rtl;
```

基本的な考え方は**図3**のように，定数で固定のフィルタ係数と可変の入力信号により得る部分積(Partial Product)を，LUTとして先に用意しておき，入力信号によってテーブルを参照することで乗算器は不要になります．後は部分積を，桁を合わせて足し合わせることでフィルタの出力値が得られます．

今回用いた8タップFIR低域通過フィルタの係数(符号付き8ビット)は次のようになります．

```
coef = [10 45 92 127 127 92 45 10];
```

これは左右対称性があるため，折り返しを用いて4タップにすることができます．4入力LUTの値($2^4 = 16$通り)を計算したのが以下のようになります．

[0, 10, 45, 55, 92, 102, 137, 147,
127, 137, 172, 182, 219, 229, 264, 274]

リスト1に，これを用いて実装したVHDL回路を紹介します．

Minseok Kim
東京工業大学